Mikael Lehtonen

Modulaarinen pelimoottori

Metropolia Ammattikorkeakoulu

Insinööri (AMK)

Tietotekniikan koulutusohjelma

Insinöörityö

26.11.2017

Metropolia

Tekijä(t) Otsikko Sivumäärä Aika	Mikael Lehtonen Modulaarinen pelimoottori 31 sivua + 2 liitettä 26.11.2017
Tutkinto	Insinööri (AMK)
Koulutusohjelma	Tietotekniikka
Suuntautumisvaihtoehto	Ohjelmistotekniikka
Ohjaaja(t)	Lehtori Miikka Mäki-Uuro

Insinöörityössä toteutettiin modulaarinen komponenttipohjainen pelimoottori ja perehdyttiin pelimoottoreiden versioihin ja historiaan sekä modernin pelimoottorin arkkitehtuuriin ja suunnitteluun. Modernin pelimoottorin arkkitehtuurista perehdyttiin komponenttipohjaisen pelimoottorin ydinkomponentteihin ja reaaliaikajärjestelmään. Komponenttipohjaisen pelimoottorin suunnittelusta perehdyttiin pelimoottorin suunnitteluperiaatteisiin ja havainnollistettiin pelisilmukan ja komponenttipohjaisen pelimoottorin suunnittelumalleja.

Insinöörityössä toteutetun pelimoottorin modulaarinen rakenne sisältää toimivan pelin toteutukseen tarvittavat ohjelmistokomponentit, joita voidaan tarvittaessa täydentää uusilla ohjelmistokomponenteilla. Ohjelmistokomponentit on jaettu ydinkomponentteihin ja peliobjektien komponentteihin sekä järjestelmästä ja sovelluksesta vastaaviin komponentteihin. Ohjelmistokomponentit ohjelmoitiin C++-ohjelmointikielellä, jota käytetään myös pelimoottorilla toteutettavien pelien ohjelmointiin. Pelimoottorin toteutuksessa hyödynnettiin suunnitteluperiaatteita ja suunnittelumalleja sekä valmiita ohjelmakirjastoja.

Avainsanat	C++, pelimoottori, modulaarisuus

Author Title Number of Pages Date	Mikael Lehtonen Modular Game Engine 31 pages + 2 appendice 26 November 2017
Degree	Bachelor of Engineering
Degree Programme	Information Technology
Specialisation option	Software Engineering
Instructor	Miikka Mäki-Uuro, Senior Lecturer

This thesis introduces the implementation of a modular component-based game engine. Different versions of game engines and their history as well as the architecture and design of a modern game engine are also introduced together with the core components and the real-time system of the modern game engine. Furthermore, the design principles of a component-based game engine and the design patterns of a game loop and a component-based game engine are discussed.

The modular design of the game engine includes the software components needed for the implementation of a video game. The modular design allows software components to be supplemented with new software components. Software components are divided into core components and components of game objects as well as system and application components. The software components are programmed with C++ programming language, which was also used for game programming in the game engine. The game engine was implemented using design principles and design templates as well as software libraries.

Keywords	C++, game engine, modularity

Sisällys

Lyhenteitä ja käsitteitä

Box2D	Avoimen lähdekoodin alustariippumaton 2D-fysiikkamoottori.
Bullet	Avoimen lähdekoodin alustariippumaton 3D-fysiikkamoottori.
DirectX	Microsoftin kehittämä ohjelmointirajapinta peliohjelmointiin.
fps	frames per second. Näyttötekniikassa ja tietokonepeleissä näytölle sekunnissa piirrettyjen kuvien, joita kutsutaan kehyksiksi, määrä.
FPS	First Person Shooter. Ampumiseen keskittyvä videopelien genre, jossa pelimaailma esitetään ensimmäisen persoonan näkökulmasta.
Havok	Kaupallisesti lisensoitava alustariippumaton fysiikkamoottori.
HID	Human interface device. Tiedon syöttämiseen tarkoitettu tietokonelaite.
HUD	Heads-up display. Näyttö, jolla pelitiedot näytetään pelinäkymän päällä.
OpenGL	Open Graphics Library. Ohjelmointirajapinta vektorigrafiikan renderöintiin.
NTSC	National Television System Committee. TV-kuvan standardi, jossa kuvataajuus on 30 Hz lomiteltuna 60 kenttään eli puolikuvaan sekunnissa.
PhysX	Ilmainen alustariippumaton fysiikkamoottori.
SDL	Simple DirectMedia Layer. Alustariippumaton ohjelmakirjasto laitteistoresurssien hallintaan.
SFML	Simple and Fast Multimedia Library. Alustariippumaton ohjelmakirjasto laitteistoresurssien hallintaan.
TPS	Third-person Shooter. Ampumiseen keskittyvä videopelien genre, jossa pelimaailma esitetään kolmannen persoonan näkökulmasta.

1 Johdanto

Pelimoottori on olennainen osa modernia pelituotantoa, jossa pelimoottoreita käytetään pelien nopeaan kehittämiseen ja valmistamiseen. Valmiiden pelimoottorien saatavuus on tehnyt pelituotannosta taloudellisesti kannattavaa myös pienille pelistudioille ja yksityisille pelikehittäjille, jotka voivat julkaista pelejä investoimatta suuria summia pelituotantoon ja pelimoottorin kehittämiseen. Suurin osa moderneista peleistä käyttää valmiita pelimoottoreita, joka on mahdollistanut merkittävän tulonlähteen kaupallisesti lisensoitavien pelimoottoreiden kehittäjille. Oman pelimoottorin kehittäminen voi olla kannattavaa, jos pelimoottorilla toteutetaan tietyn pelityypin pelejä tietyille alustoille tai pelimoottori lisensoidaan kaupallisesti myös muille pelikehittäjille. Oman pelimoottorin toteuttaminen mahdollistaa pelimoottorin tehokkaan optimoinnin, joka voi olla huomattavasti helpompaa verrattuna kaupallisesti lisensoitaviin pelimoottoreihin.

Insinöörityön tavoitteena oli toteuttaa modulaarinen komponenttipohjainen pelimoottori, joka sisältää toimivan pelin toteutukseen tarvittavat ohjelmistokomponentit. Modulaarinen rakenne mahdollistaa pelimoottorin täydentämisen uusilla ohjelmistokomponenteilla ja tarvittaessa ohjelmistokomponenttien vaihtamisen ominaisuuksiltaan samankaltaisiin ohjelmistokomponentteihin. Tavoitteena oli myös perehtyä modernin komponenttipohjaisen pelimoottorin arkkitehtuuriin ja suunnitteluperiaatteisiin sekä modernin pelimoottorin suunnittelussa ja toteutuksessa käytettäviin suunnittelumalleihin.

Insinöörityöraportti sisältää seitsemän lukua, joissa käsitellään pelimoottorin merkitystä pelituotannossa sekä modernin pelimoottorin kehitykseen ja toteutukseen liittyviä asioita. Raportin alkuosassa tutustutaan pelimoottoreiden historiaan ja versioihin sekä modernin komponenttipohjaisen pelimoottorin arkkitehtuuriin ja ydinkomponentteihin. Puolivälissä havainnollistetaan suunnitteluperiaatteiden ja suunnittelumallien soveltamista modernin pelimoottorin suunnittelussa ja toteutuksessa. Raportin loppuosassa perehdytään insinöörityössä toteutetun komponenttipohjaisen pelimoottorin arkkitehtuuriin ja komponentteihin sekä toteutetun pelimoottorin testaukseen ja analysointiin.

2 Modernit pelimoottorit

Pelimoottori on ohjelmistokehys, joka on tarkoitettu pelien kehittämiseen ja valmistamiseen. Moderni pelimoottori sisältää pelin toteuttamisessa tarvittavat ohjelmistokomponentit ja usein myös pelin sisältöön ja kehitykseen liittyviä työkaluja. Pelimoottori toimii usein pelikehityksen keskipisteenä, jonka ympärillä tuotantotiimi työskentelee. [1, s. 4.]

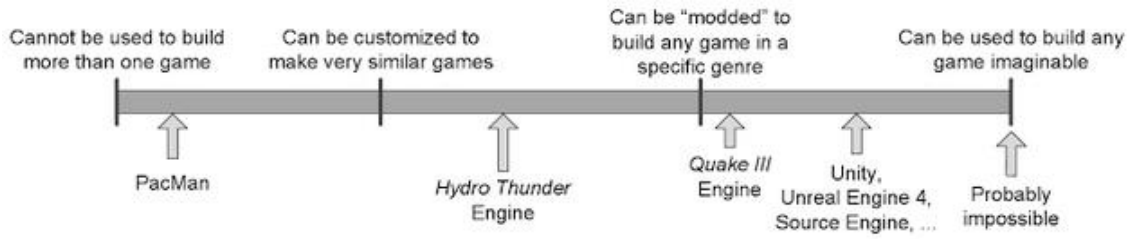
2.1 Historia

Modernien pelimoottorien historian katsotaan alkaneen 1990-luvun puolivälissä, jolloin pelimoottoria käytettiin terminä viittaamaan suosittujen id Softwaren kehittämien Doom ja Quake FPS-pelien samankaltaiseen ohjelmistoarkkitehtuuriin, jossa pelin ydinkomponentit oli eroteltu toisistaan. Erottelu mahdollisti pelien lisensoinnin ja uudelleentyöstämisen uusiksi tuotteiksi ainoastaan pienillä ydinkomponenttien muutoksilla, tarvitsematta muuttaa koko pelin rakennetta. Tämä mahdollisti itsenäisten pelikehittäjien ja pienien pelistudioiden valmistaa uusia pelejä muokkaamalla olemassa olevia pelejä. 1990-luvun loppupuolella osa pelistä, kuten Quake III Arena ja Unreal suunniteltiin uudelleenkäytettäviksi ja -muokattaviksi. Uusista pelimoottoreista tehtiin entistä muokattavampia komentosarjakielillä ja pelimoottorien lisensointi alkoi olla kannattava toissijainen tulovirta niiden kehittäjille. Nykypäivän pelikehittäjät voivat lisensoida pelimoottorin ja käyttää merkittävän osan sen tarjoamista ohjelmistokomponenteista valmistaakseen uuden pelin. Vaikka pelimoottorin lisensointi sisältää edelleen huomattavia investointeja pelituotantoon, se on usein paljon taloudellisempaa kuin kehittää pelimoottori ja kaikki sen tarvitsemat komponentit itse. [2, s. 11.]

2.2 Versiot

Pelimoottoreista on julkaistu useita versioita, joista useimmat on tehty tietyille peleille ja laitteistoalustoille. Yleiskäyttöisillä pelimoottoreilla voidaan kehittää useita erityyppisiä pelejä, mutta todellisuudessa myös monet yleiskäyttöiset pelimoottorit ovat sopivia vain tietyille pelityypeille. Yleiskäyttöiset pelimoottorit ovat yleensä vähemmän optimaalisia ajamaan tiettyä pelityyppiä tietyllä alustalla. Peleistä voidaankin tehdä vaikuttavampia optimoimalla pelimoottori pelin ja/tai laitteistoalustan vaatimuksille ja rajoitteille. [2, s. 12.]

Toisen persoonan taistelupeleille kehitetty pelimoottori on hyvin erilainen kuin valtavan online-moninpelin, ensimmäisen persoonan ammuntapelin tai reaaliaikaisen strategiapelin pelimoottori. Pelimoottoreissa on kuitenkin paljon päällekkäisyyksiä, esimerkiksi kaikki 3D-pelit, riippumatta pelityypistä, edellyttävät syötteitä pelaajalta, jonkin muotoista 3D-mallinnusta, HUD-näytön pelitiedoille, äänentoistojärjestelmän ja paljon muita samankaltaisuuksia. Esimerkiksi vaikka Unreal Engine -pelimoottori suunniteltiin ensimmäisen persoonan ammuntapeleille, sitä on käytetty menestyksekkäästi myös monien muiden pelityyppien pelien tekemiseen (kuva 1). [2, s. 12.]



Kuva 1. Pelimoottorien uudelleenkäytettävyys. [1, s. 12.]

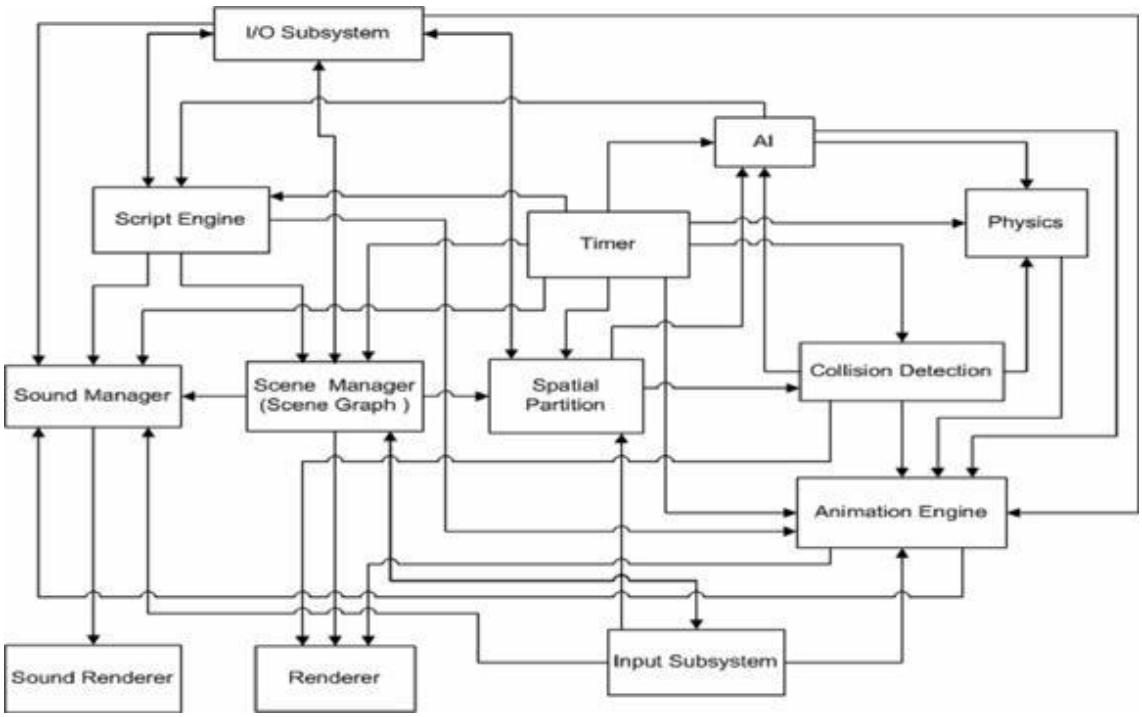
Taulukossa 1 on muutamia tunnettuja pelimoottoreita, joista ensimmäisiä kaupallisesti lisensoitavia pelimoottoreita olivat id Softwaren Quake-pelimoottorit (1992). Quake-pelimoottoreita käytettiin varhaisissa FPS-peleissä. Id Software on julkaissut Quake-pelimoottorista myös uusia id Tech -versioita, joita on käytetty uudemmissa id Softwaren ammuntapeleissä. Kaupallisesti lisensoitavista pelimoottoreista uudempia ovat Epic Gamesin Unreal -pelimoottorit (1998), joita on käytetty monissa moderneissa FPS- ja TPS-peleissä. Unreal-pelimoottorin uudemmilla versioilla on mahdollista kehittää myös muiden pelityyppien pelejä. Uusista yleiskäyttöisistä pelimoottoreista suosituimpia on Unity-pelimoottori (2005), jota on käytetty useissa moderneissa tietokone-, konsoli- ja mobiilipeleissä. Avoimenlähdekoodin pelimoottoreista tunnettuja ovat Panda3D ja Irrlicht, jotka soveltuvat hyvin pieniin projekteihin ja prototyyppien toteuttamiseen. [2, s. 25.]

Taulukko 1. Tunnettuja pelimoottoreita. [3]

Moottori	**Ohjelmointi**	**Skriptikieli**	**2D/3D**	**Alustat**	**Lisenssi**
id Tech 4	C++	C++	3D	Windows, Linux, OS X	GPL
Irrlicht	C++		2D, 3D	Windows, Linux, OS X	zlip
Panda3D	C++, Python	Python	2D, 3D	Windows, Linux, OS X	BSD
Unity	C, C++, C#, Java	UnityScript, Boo, C#, Cg	2D, 3D	Cross-platform	Kaupallinen
Unreal	C++, C#, Assembly	GLSL, Cg, UnrealScript	2D, 3D	Cross-platform	Kaupallinen

3 Pelimoottorien arkkitehtuuri

Modernin pelimoottorin arkkitehtuuri sisältää yleensä laajan joukon komponentteja (kuva 2), joilla on mahdollista kehittää useita erityyppisiä pelejä. Pelimoottorin ominaisuudet on jaettu ydinkomponentteihin, joista jokainen on vastuussa erilaisista toisiinsa liittyvistä tehtävistä. Esimerkiksi fysiikan lakeja soveltava ydinkomponentti on vastuussa siitä, että painovoima vetää ilmassa olevia esineitä alaspäin ja että objektien väliset törmäykset havaitaan. Pelimoottori sisältää usein myös komponentteja, jotka monipuolistavat pelimoottorin ominaisuuksia ja mahdollistavat erityyppisten pelien kehittämisen. Ominaisuuksien jakaminen komponentteihin on hyödyllistä ohjelmoijille, koska se sallii pelimoottorin lähdekoodin jakamisen useisiin komponentteja vastaaviin tiedostoihin ja mahdollistaa koodin hallitsemisen komponenttien toiminnallisuuden mukaan sekä nopeuttaa testaamista ja suorituksenaikaisten virheiden jäljittämistä. Ydinkomponentit toteutetaan yleensä singleton-mallin mukaisesti, jolloin voidaan varmistua, ettei ydinkomponentista ole suorituksenaikana muistissa enempää kuin yksi konkreettinen ilmentymä. [1, s. 9.]



Kuva 2. Modernin pelimoottorin komponentteja. [4]

3.1 Ydinkomponentit

Suurin osa pelimoottoreista sisältää toimivan pelin kannalta olennaiset ydinkomponentit, joita ovat pelimaailman renderöintiin ja hallintaan sekä pelilogiikkaan ja pelaamiseen liittyvät ydinkomponentit. Pelimoottorin ominaisuudet on jaettu pelimoottorin resursseista ja toiminnasta vastaaviin ydinkomponentteihin, jotka ovat vastuussa erilaisista toisiinsa liittyvistä tehtävistä. [1, s. 18.]

Resurssienhallinta

Resurssienhallinta tarjoaa yhtenäisen rajapinnan tai rajapintoja kaikkien ja kaiken tyyppisten peliresurssien hallintaan. Resurssienhallinta voidaan toteuttaa keskitetysti resurssienhallintakomponentilla, tai jättää tapauskohtaisesti peliohjelmoijan toteutettavaksi, jolloin peliohjelmoija voi ladata tiedostot suoraan massamuistista tai pelimoottorikohtaisista arkistotiedostoista. [2, s. 35.]

Resurssit ovat pelissä käytettävää digitaalista materiaalia, jotka voivat olla mediatiedostoja ja skriptitiedostoja. Mediatiedostoilla toteutetaan pelin visuaalinen ja auditiivinen sisältö. Skriptitiedostot ovat tiedostoja, joilla määritellään pelin käyttäytyminen ja toiminta suorituksenaikana. Resurssitiedostot sijaitsevat yleensä massamuistissa ja voivat olla mediatiedostojen osalta kuva-, audio- tai videotiedostoja ja käyttäytymistiedostojen osalta teksti- tai skriptitiedostoja. [1, s. 10.]

Resurssienhallintakomponentin tehtävänä on hallita pelimoottorin ja resurssien välistä suhdetta. Resurssienhallintakomponentti tunnistaa ja erottaa saatavilla olevat resurssit, lataa resurssit tiedostoista muistiin ja purkaa ladatut resurssit muistista sekä varmistaa, ettei muistiin ole ladattu useita ilmentymiä samoista resursseista. [1, s. 11.]

Renderöintimoottori

Renderöintimoottori on yksi suurimmista ja monimutkaisimmista pelimoottorin ydinkomponenteista. Renderöintimoottorit voivat olla arkkitehtuuriltaan hyvin erilaisia, vaikka suurin osa renderöintimoottoreista jakaa joitain perusominaisuuksia, jotka perustuvat suurelta osin grafiikkalaitteiston suunnitteluun, josta ne ovat riippuvaisia. [2, s. 36.]

Renderöintimoottorin tehtävänä on hakea grafiikkaresurssit muistista ja piirtää resurssit grafiikkalaitteiston kautta näytölle. Renderöintimoottori päättää, kuinka resurssin pikselit piirretään näytölle ja miten pikseleistä muodostuva kuva asetellaan näytöllä. Renderöinti prosessista vastaa grafiikkakirjasto, kuten OpenGL, DirectX tai SDL ja mikä tahansa näistä yhdessä peli- ja renderöintimoottoriin ohjelmoitujen toimintojen. [1, s. 11.]

Fysiikkamoottori

Fysiikkamoottorin tehtävänä on soveltaa fysiikan lakeja peliobjekteihin. Fysiikkamoottori hallitsee peliobjektien suorituksenaikaista käytöstä ja on vastuussa voimien, vääntömomenttien, painovoiman ja inertian soveltamisesta peliobjekteihin sekä peliobjektien välisen törmäyksen havaitsemisesta ja vaikutuksesta. [1, s. 14.]

Moderneissa pelimoottoreissa fysiikka on tarkemmin kuvattuna jäykän kappaleen dynamiikan simulointia. Jäykän kappaleen simulointi vähentää törmäyksen havaitsemisessa tarvittavan laskennan määrää. Fysiikkamoottori voi olla merkittävä osuus pelimoottorin lähdekoodista, joten monesti pelimoottorin yhteydessä käytetään valmista fysiikkamoottoria, kuten Bullet, PhysX tai Havok. [2, s. 602.]

Syöttöjärjestelmä

Syöttöjärjestelmä prosessoi pelaajan antamia syötteitä, jotka on saatu erilaisilta HID-laitteilta (human interface devices), joilla pelaaja voi vaikuttaa pelin tapahtumiin. Pelissä käytettäviä HID-laitteita voivat olla esimerkiksi näppäimistö, hiiri, peliohjain tai muut peleihin tarkoitetut ohjaimet. Syöttöjärjestelmä toteutetaan yleensä kaksisuuntaisesti, jolloin pelaajalle voidaan antaa palautetta pelin tapahtumista. Palaute annetaan siihen erikoistuneilta HID-laitteilta, joita voivat olla haptisilla tai auditiivisilla toiminnoilla varustetut peliohjaimet. Syöttöjärjestelmän toimintona on usein painikeasettelujen ohjelmallinen muuttaminen sekä yhtäaikaisen, jaksottaisen ja järjestyksessä tapahtuvan painamisen havaitseminen. [2, s. 36.]

Audiojärjestelmä

Audiojärjestelmä vastaa auditiivisista resursseista ja niiden toistamisesta äänentoistolaitteistolla. Pelissä auditiiviset resurssit ovat musiikkia tai ääntä. Musiikki on pitempiä äänitiedostoja, joita toistetaan toistuvasti pelin taustalla. Äänet ovat lyhyitä äänitiedostoja, joita toistetaan pelitapahtumien yhteydessä. Audiojärjestelmä on vastuussa myös äänitasojen asettamisesta ja ääneen vaikuttavien tehosteiden toteutumisesta. [1, s. 12.]

Näkymänhallinta

Näkymänhallinta lataa pelinäkymän ja yhdistää pelinäkymän resurssit ja ydinkomponenttien toiminallisuuden pelattaviksi tapahtumiksi. Näkymänhallinta hallitsee pelinäkymän peliobjekteja ja tarjoaa pääsyn peliobjekteihin pelinäkymän ulkopuolelta. Pelinäkymät ovat ladattavia tiedostoja, jotka sisältävät pelissä käytettävien resurssien asettelun ja toiminnallisuuden sekä mahdollisuuden vaikuttaa pelinäkymän tapahtumiin. [1, s. 14.]

Skriptijärjestelmä

Skriptikieltä käytetään pelimoottorissa nopeuttamaan ja helpottamaan pelimoottorilla tehtävää pelikehitystä ja peliohjelmointia. Skriptikieltä käytettäessä pelin suoritustiedostoja ei tarvitse kääntää ja linkittää uudelleen tehtäessä muutoksia pelin pelilogiikkaan ja tietorakenteeseen. Skriptijärjestelmä mahdollistaa pelilogiikan ja tietorakenteen muuttamisen ja uudelleenlataamisen suorituksen tai sammuttamisen yhteydessä. [2, s. 47.]

Viestijärjestelmä

Viestijärjestelmän tehtävänä on välittää viestejä peliobjektien välillä. Viestijärjestelmä toteutetaan usein objektien väliseen viestintään tarkoitetulla tapahtumapohjaisella arkkitehtuurilla. Tapahtumapohjaisessa arkkitehtuurissa viestin lähettävä objekti luo datatietueen, joka sisältää viestityypin ja lähetettävän datan. Viesti välitetään kutsumalla viestin vastaanottavan objektin tapahtumankäsittelijäfunktiota. Viesti voidaan myös asettaa jonoon ja välittää myöhemmin vastaanottavalle objektille. [2, s. 47.]

3.2 Reaaliaikajärjestelmä

Videopelit ovat interaktiivisia ajallisesti simuloituja agentti-pohjaisia pehmeitä reaaliaikajärjestelmiä, joissa pelaajalla on mahdollisuus vaikuttaa interaktiivisesti pelin tapahtumiin ja erillisiin agentti-pohjaisiin yksiköihin, kuten peliobjekteihin, jotka ovat vuorovaikutuksessa keskenään virtuaalisessa dynaamisesti muuttuvassa pelimaailmassa. Pehmeissä reaaliaikajärjestelmissä järjestelmän poikkeuksellinen toiminta tai asetetuista aikarajoista poikkeaminen ei pysäytä järjestelmän suoritusta. [2, s. 9.]

Reaaliaikajärjestelmä noudattaa aikarajoja. Esimerkiksi videopelin näyttöä täytyy päivittää vähintään 24 kertaa sekunnissa, jotta liikkeen illuusio saadaan aikaan. Suurin osa videopeleistä päivittää näyttöä 30 tai 60 kertaa sekunnissa, koska nämä ovat NTSC-monitorin virkistystaajuuden monikertoja. Fysiikan vakaa simulointi saattaa tarvita päivittämistä 120 kertaa sekunnissa, älykkäältä vaikuttava tekoäly on päivitettävä vähintään kerran sekunnissa ja äänen virheetön toiminta voi edellyttää äänikirjaston kutsumista ainakin 60 kertaa sekunnissa. [2, s. 10.]

3.2.1 Kehysnopeus

Reaaliaikaisen pelin kehysnopeus kertoo, kuinka nopeasti kuvien, joita kutsutaan kehyksiksi, sarja esitetään pelaajalle. Peleissä ja elokuvissa kehysnopeus ilmoitetaan tyypillisesti kehyksiä sekunnissa (fps). Elokuvissa kehysnopeus on perinteisesti 24 fps. Peleissä kehysnopeus on NTSC-standardin mukaisesti 30 tai 60 fps. Kehysten välillä kuluva aika tunnetaan kehysaikana tai delta-aikana, joista delta-aika on yleisempi, koska kehysten välillä kuluva aika esitetään usein matemaattisesti symbolilla Δt. Teknisesti Δt kutsutaan kehysjaksoksi, koska se on käänteinen kehystaajuudesta $T = 1 / f$, mutta peleissä ”jakso” -termiä ei käytetä tässä yhteydessä. Jos pelin kehysnopeus on 30 fps, niin pelin delta-aika on 1/30 sekuntia, tai 33,3 millisekuntia. Kehysnopeudella 60 fps pelin delta-aika on puolet pienempi, eli 1/60 sekuntia tai 16,6 millisekuntia. [2, s. 312.]

Peleissä käytetyt numeeriset integraatiomenetelmät, kuten Eulerin menetelmä, toimivat hyvin, kunhan peliobjektien nopeudet ovat suunnilleen vakioita. Peliobjektien havaitut nopeudet ovat riippuvaisia kehysten välillä kuluneesta ajasta, jota voidaan käyttää peliobjektien liikuttamiseen vakionopeudella. Vakionopeudella liikkuvan peliobjektin kehysaikana kulkema matka lasketaan kertomalla peliobjektin nopeus delta-ajalla. [2, s. 313.]

3.2.2 Pelisilmukka

Numeeriset simulaatiot toteutetaan tyypillisesti suorittamalla laskutoimituksia toistuvasti kullekin diskreetille ajanhetkelle. Videopelit ja numeeriset simulaatiot toimivat samalla tavalla. Videopeleissä pelisilmukkaa suoritetaan toistuvasti ja jokaisen iteraation aikana pelijärjestelmille annetaan mahdollisuus laskea tai päivittää tilansa seuraavaan diskreettiin ajanhetkeen. [2, s. 9.]

Pelisilmukka voidaan toteuttaa useilla eri tavoilla, mutta niiden toteutus sisältää yleensä yhden tai useita toistuvasti päivittyviä silmukoita, joissa on erilaisia toiminnallisuuksia. Windows-ympäristössä pelisilmukan tarvitsee pelijärjestelmien lisäksi palvella Windows-viestejä ja tarvittaessa pysäyttää ja palauttaa pelijärjestelmien päivitys viestien ilmetessä. Pelijärjestelmien toteutuksessa käytettävät ohjelmakirjastot saattavat myös tarvita erityisjärjestelyjä pelisilmukan toteutuksessa. [2, s. 308.]

4 Pelimoottorien suunnittelu

Komponenttipohjaisten pelimoottoreiden suunnittelu sisältää hyvin samankaltaisia ratkaisuja, joissa pelimoottorin ominaisuudet on jaettu toisiinsa liittyviin ydinkomponentteihin, joita päivitetään pelimoottorin pelisilmukassa. Pelimoottorien samankaltainen suunnittelu mahdollistaa hyväksi todettujen suunnitteluperiaatteiden ja -mallien hyödyntämisen komponenttipohjaisten pelimoottoreiden suunnittelussa. [1, s. 20.]

4.1 Suunnitteluperiaatteet

Suunnitteluperiaatteet ovat hyväksi todettuja periaatteita, joilla voidaan välttää keskeisiä suunnitteluongelmia. Nykyaikaisessa pelimoottorisuunnittelussa on useita suunnitteluperiaatteita, joita on käytetty monen menestyksekkään pelimoottorin suunnittelussa ja toteutuksessa. Neljä hyväksi todettua pelimoottorin suunnitteluperiaatetta ovat kierrätettävyys, käsitteellisyys, modulaarisuus ja yksinkertaisuus. [1, s. 20.]

Kierrätettävyys

Kierrätettävyys on pelimoottorin suunnitteluperiaate, jossa ladattuja resursseja käytetään suorituksenaikana uudelleen. Ladattujen resurssien uudelleenkäyttö parantaa pelimoottorin suorituskykyä, koska resurssienhallinnan ei tarvitse ladata samoja resursseja toistamiseen. Suorituskyvyn voidaan katsoa parantuneen aina, kun ladattuja resursseja on käytetty onnistuneesti ja luotettavasti uudelleen. [1, s. 20.]

Käsitteellisyys

Käsitteellisyys on pelimoottorin suunnitteluperiaate, jossa tuotetaan yleistyksiä tietyistä ilmentymistä. Esimerkiksi pelimoottorin ydinkomponentit ovat käsitteellisiä, jos niiden tarkkoja yksityiskohtia ei ole määritelty ja ne eivät ole riippuvaisia mistään yksittäisistä tilanteista. Käsitteellisyydestä on hyötyä ydinkomponenttien suunnittelussa, koska käsitteellisyys lisää niiden käytettävyyttä ja monipuolisuutta. [1, s. 21.]

Modulaarisuus

Modulaarisuus on pelimoottorin suunnitteluperiaate, jossa pelimoottorin ominaisuudet jaetaan tarkoituksen tai tehtävän mukaisiin ydinkomponentteihin. Ydinkomponentit ovat itsenäisesti toimivia yksiköitä, jotka vastaavat erilaisista pelimoottorin tehtävistä. Itsenäisesti toimivat yksiköt eroavat toiminnaltaan muista yksiköistä ja ovat vaihdettavissa ominaisuuksiltaan samankaltaisiin yksiköihin. Pelimoottorin modulaarisuus mahdollistaa monimutkaisten kokonaisuuksien yksinkertaistamisen komponentteihin ja muutosten tekemisen komponenttiin vaikuttamatta muihin komponentteihin. [1, s. 22.]

Yksinkertaisuus

Yksinkertaisuus on pelimoottorin suunnitteluperiaate, joka varmistaa, että pelimoottorin ydinkomponentit eivät sisällä samoja tai hyvin samankaltaisia ominaisuuksia. Yksinkertaisuuden suunnitteluperiaatetta käytetään modulaarisen pelimoottorin ydinkomponenttien toteuttamisessa, jossa pelimoottorin monimutkaiset kokonaisuudet yksinkertaistetaan jakamalla samankaltaiset ominaisuudet erilaisiin ydinkomponentteihin. [1, s. 23.]

4.2 Suunnittelumallit

Suunnittelumallit ovat hyväksi todettuja ratkaisuja yleisesti tunnettuihin oliopohjaisen ohjelmistosuunnittelun tilanteisiin. Suunnittelumalli sisältää ratkaistavan tilanteen kuvauksen ja esimerkin tilanteen ratkaisusta. Pelisuunnittelussa on oma joukko erilaisia pelisuunnittelumalleja, joilla voidaan ratkaista pelisuunnitteluun liittyviä tilanteita. [2, s. 96.]

4.2.1 Pelisilmukka

Pelisilmukka on pelisuunnittelun keskeinen suunnittelumalli, jossa tarkoituksena on päivittää ja renderöida pelitapahtumat toistuvasti päivittyvässä silmukassa sekä käsitellä pelaajan syötteet ja seurata kulunutta reaaliaikaa pelinopeuden hallitsemiseksi. Pelisilmukka voidaan suunnitella toteutettavaksi kiinteällä kehysnopeudella ja interpoloidulla renderoinnilla (esimerkkikoodi 1), joka mahdollistaa pelijärjestelmien reaaliaikaisen päivittämisen ja peligrafiikan tasaisemman renderöinnin. [5.]

```
double previous = getCurrentTime();
double lag = 0.0;
while (true)
{
    double current = getCurrentTime();
    double elapsed = current - previous;
    previous = current;
    lag += elapsed;

    processInput();

    while (lag >= MS_PER_UPDATE)
    {
        update();
        lag -= MS_PER_UPDATE;
    }

    render(lag / MS_PER_UPDATE);
}
```

Esimerkkikoodi 1. Interpoloitu pelisilmukka. [5]

Interpoloidussa pelisilmukassa (esimerkkikoodi 1) lasketaan silmukan edellisestä kierroksesta kulunutta reaaliaikaa, joka kertoo, kuinka paljon peliaikaa on simuloitava kiin-

teän kehysnopeuden saavuttamiseksi. Kuluneella reaaliajalla päivitetään peliajan viivettä, jonka täyttyessä pelijärjestelmät päivitetään pelisilmukan sisemmässä silmukassa toistuvasti, kunnes kiinteä kehysnopeus on saavutettu. Kiinteä kehysnopeus tekee pelijärjestelmien toiminnasta vakaampaa ja luotettavampaa. Kehysnopeuden muutokset eivät vaikuta renderöintimoottorin toimintaan, joka renderöi peligrafiikan muutosten tapahtuessa, joten peligrafiikan renderöinti toteutetaan aina, kun se on mahdollista. [5.]

4.2.2 Päivitysmalli

Päivitysmalli on pelisuunnittelumalli, jossa pelisilmukassa päivitettävien objektien päivitys toteutetaan pelisilmukasta kutsuttavilla päivitysfunktioilla (esimerkkikoodi 2). Päivitysfunktioilla toteutettava objektien päivitys selkeyttää pelisilmukan rakennetta ja mahdollistaa objektien helpon lisäyksen ja poistamisen. Päivitysmallissa pelisilmukka päivittää ainoastaan polymorfisten objektien kokoelmaa. Päivitysmalli erottaa päivitettävien objektien käyttäytymisen pelisilmukasta ja muista päivitettävistä objekteista. Päivitysmalli on hyödyllinen peleissä, joissa on useita toisistaan riippumattomia objekteja tai järjestelmiä, jotka täytyy suorittaa toistuvasti ja samanaikaisesti. [6.]

```
void GameLoop()
{
    while (true)
    {
        // Handle user input...

        // Update each entity.
        for (int i = 0; i < numEntities_; i++)
        {
            entities_[i]->update();
        }

        // Physics and rendering...
    }
}
```

Esimerkkikoodi 2. Päivitysmallin pelisilmukka. [6]

Päivitysmallin pelisilmukka (esimerkkikoodi 2) päivittää päivitysfunktion sisältämät objektit iteroimalla objektien kokoelmaa ja kutsumalla objektien päivitysfunktioita. Objektien päivitysfunktiot voidaan toteuttaa objektien yhteisestä kantaluokasta perittävällä abstraktilla päivitysfunktiolla, jonka pelisilmukassa päivitettävä objekti ylikirjoittaa. [6.]

4.2.3 Komponentti

Komponenttimalli on pelisuunnittelumalli, jossa ideana on pakata peliobjektien toiminnallisuus erillisiin objekteihin ja välttää luokkien perinnästä aiheutuvia luokkahierarkioita. Komponenttimallissa peliobjekti on kokoelma toiminnaltaan erilaisia komponentteja, jotka yhdessä määrittävät peliobjektin käyttäytymisen, ulkoasun ja toiminnallisuuden. [7.]

Komponenttimallin pelimoottori (esimerkkikoodi 3) rakentuu pelisilmukasta ja ydinkomponenteista (esimerkkikoodi 4), jotka sijaitsevat pelimoottorin tietorakenteessa. Pelisilmukka on toistuvasti päivittyvä silmukka, joka päivittää ydinkomponentit kutsumalla pelimoottorin sisäistä päivitysfunktiota (esimerkkikoodi 5), joka iteroi pelimoottorin tietorakenteen ja kutsuu ydinkomponenttien sisäisiä päivitysfunktioita (esimerkkikoodi 6). [7.]

```
class Engine
{
public:
    void Update(float dt);
    void GameLoop(void);
    void Add(System *sys);
private:
    std::vector<System> m_systems;
};
```

Esimerkkikoodi 3. Komponenttimallin pelimoottori. [7]

Komponenttimallin ydinkomponentit (esimerkkikoodi 4) vastaavat erilaisista pelimoottorin toiminnallisuuksista ja peliobjekteihin liittyvistä komponenteista sekä komponenttien päivittämisestä. Esimerkiksi renderöintimoottori vastaa spriteihin ja grafiikkaan liittyvistä komponenteista ja spriteihin ja grafiikkaan liittyvien komponenttien päivittämisestä. [7.]

```
class System
{
public:
    virtual void Init(void) = 0;
    virtual void Update(float dt) = 0;
    virtual void SendMessage(Message *msg) = 0;
    virtual ~System() {}
};
```

Esimerkkikoodi 4. Komponenttimallin ydinkomponentti. [7]

Ydinkomponenttien päivitysfunktio (esimerkkikoodi 5) on pelimoottorin (esimerkkikoodi 3) pelisilmukasta kutsuttava funktio, joka iteroi pelimoottorin tietorakenteen ydinkomponentit (esimerkkikoodi 4) ja kutsuu ydinkomponenttien päivitysfunktioita, joille välitetään parametreina ydinkomponenttiin liittyvät komponentit ja pelisilmukan delta-aika. [7.]

```
void Engine::Update(float dt)
{
    for (unsigned i = 0; i < m_systems.size(); ++i)
    {
        m_systems[i].Update(dt, ObjectFactory->GetObjectList());
    }
}
```

Esimerkkikoodi 5. Ydinkomponenttien päivitysfunktio. [7]

Ydinkomponentin päivitysfunktio (esimerkkikoodi 6) päivittää ydinkomponenttiin liittyvien komponenttien päivitysfunktiot. Ydinkomponenttien päivitysfunktiot päivitetään pelisilmukan päivitysfunktiolla (esimerkkikoodi 5), joka kutsuu ydinkomponenttien päivitysfunktioita (esimerkkikoodi 5). Ydinkomponentin päivitysfunktio saa argumentteina ydinkomponentissa päivitettävien komponenttien peliobjektit, joiden komponentit päivitetään kutsumalla komponenttien päivitys funktioita, joille välitetään pelisilmukan delta-aika. [7.]

```
void SomeSystem::Update(float dt, GameObjectList objects)
{
    while (objects)
    {
        SomeComponent *comp = objects->GetComponent(id);
        comp->DoStuff(dt);
        // ...

        objects = objects->next;
    }
}
```

Esimerkkikoodi 6. Ydinkomponentin päivitysfunktio. [7]

Komponenttimallin peliobjektin (esimerkkikoodi 7) komponentit sijaitsevat yleensä peliobjektin toteuttamassa tietorakenteessa, joka pitää komponentit samassa paikassa muistissa. Peliobjektien komponentit päivitetään tavallisesti ydinkomponentin päivitysfunktiossa (esimerkkikoodi 5), joka iteroi järjestyksessä kaikkien peliobjektien kom-

ponentit. Komponenttien sijoittaminen erillisiin tietorakenteisiin parantaa päivityksen tehokkuutta ja lyhentää päivitykseen kuluvaa aikaa. Komponenttien tulisikin sijaita komponentin päivityksestä vastaavan ydinkomponentin tietorakenteessa, josta komponentit päivitetään lineaarisesti iteroimatta erikseen jokaisen peliobjektin komponentteja. [7.]

```
class GameObject
{
public:
    Component *GetComponent(id);
    void AddComponent(Component *comp);
    bool HasComponent(id);
private:
    std::vector<Component *> m_components;
};
```

Esimerkkikoodi 7. Komponenttimallin peliobjekti. [7]

Indeksoitu peliobjekti (esimerkkikoodi 8) on pelimoottorin tietorakenteeseen sijoitettava peliobjekti, joka toteutetaan indeksoitavalla tietueella, jonka toteutuksessa tietorakenteeseen viitataan peliobjektin indeksillä. Indeksoitu peliobjekti sisältää myös tietorakenteen peliobjektiin liittyvien komponenttien indekseille, jotka vastaavat komponentin indeksejä komponentista vastaavan ydinkomponentin tietorakenteessa. [7.]

```
struct GameObject
{
    // index location
    unsigned m_handle;

    // Array of component handles
    unsigned m_components[eNumComponents];
};
```

Esimerkkikoodi 8. Indeksoitu peliobjekti. [7]

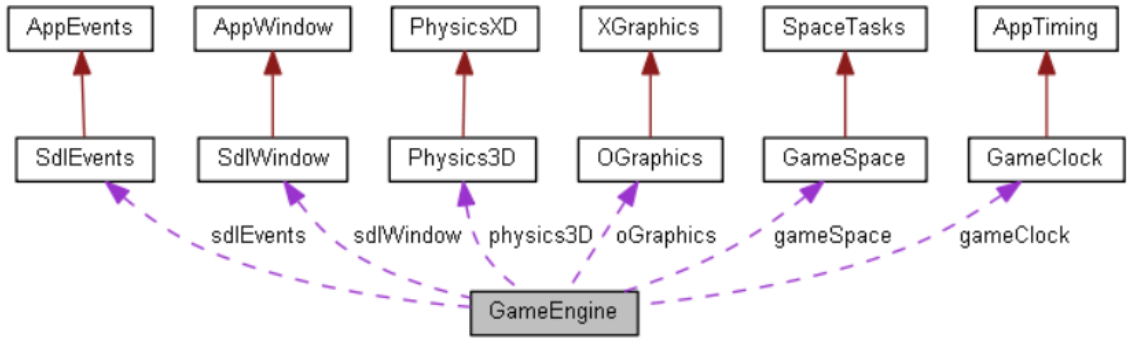
Komponenttien viestintä (esimerkkikoodi 9) tarkoittaa komponenttien välistä viestintää, jossa toimitetaan tieto tapahtumasta tai muutoksesta. Viestintä voidaan toteuttaa viestin lähettävällä ja vastaanottavalla funktiolla sekä viestin välittävällä tietueella. Viesti toimitetaan kaikille viestin vastaanottaville komponenteille, joiden viestintä funktiossa tarkistetaan viestin tunniste ja toimitaan tarvittaessa tunnisteen edellyttämällä tavalla. [7.]

```
struct TreasureChest : public LogicComponent
{
    virtual void SendMessage(MSG *msg);
};

void TreasureChest::SendMessage(MSG *msg)
{
    switch (msg->id)
    {
    case ePlayerNearby:
        this->Glimmer((LocationMSG *)msg);
        break;
    case eActivate:
        this->Open();
        break;
    }
}

struct LocationMSG
{
    enum id;
    Vec2 position;
};
```

Esimerkkikoodi 9. Komponenttien viestintä. [7]

5 Toteutus ja rakenne

Insinöörityössä toteutettiin modulaarinen pelimoottori, jonka rakenne sisältää toimivan pelin toteuttamiseen tarvittavat komponentit. Pelimoottori on ohjelmoitu C++-ohjelmointikielellä, jota käytetään myös pelimoottorilla toteutettavien pelien ohjelmointiin. Ohjelmistokomponentit on jaettu ydinkomponentteihin ja peliobjektien komponentteihin sekä järjestelmästä ja sovelluksesta vastaaviin komponentteihin. Pelimoottorin ydinkomponentit (kuva 3) toteuttavat abstraktin rajapinnan, joka mahdollistaa ydinkomponenttien nopean ja helpon vaihtamisen toiminnaltaan ja ominaisuuksiltaan samankaltaisiin ydinkomponentteihin. Abstraktit rajapinnat sisältävät ydinkomponenteissa toteutettavien palveluiden virtuaaliset rajapintafunktiot ja osoittimet ydinkomponentissa toteutettuihin palveluihin. Osoittimet mahdollistavat palveluiden käyttämisen tuntematta palveluita toteuttavien ydinkomponenttien toteutusta.



Kuva 3. Pelimoottorin ydinkomponentit.

Pelimoottorin toteutus sisältää seitsemän ydinkomponenttia (kuva 3), joista keskeisin on ydinkomponenttien alustuksesta ja pelisilmukan toteutuksesta vastaava GameEngine-ydinkomponentti. Sovellus- ja järjestelmätapahtumista vastaa SdlEvents-ydinkomponentti, joka toteuttaa AppEvents-rajapinnan ja jonka palveluita käyttävät SdlWindow-ydinkomponentti ja syöttöjärjestelmästä vastaava AppInputs-komponentti. Sovellusikkunasta ja vastaa SdlWindow-ydinkomponentti, joka toteuttaa AppWindow-rajapinnan ja jonka palveluita käyttää grafiikan renderöinnista vastaava OGraphics-ydinkomponentti. Fysiikan simuloinnista ja törmäyksien havaitsemisesta vastaa Physics3D-ydinkomponentti, joka toteuttaa PhysicsXD-rajapinnan ja jonka palveluita käyttävät peliobjektien fysiikasta vastaavat komponentit. Peligrafiikan renderöinnista vastaa OGraphics-ydinkomponentti, joka toteuttaa XGraphics-rajapinnan ja jonka palveluita käyttävät peliobjektien renderöinnista vastaavat komponentit. Näkymänhallinnasta vastaa GameSpace-ydinkomponentti, joka toteuttaa SpaceTasks-rajapinnan ja jonka palveluita käyttävät näkymänhallintaan vaikuttavat komponentit. Pelisilmukan ajoituksesta ja kehysnopeudesta vastaa GameClock-ydinkomponentti, joka toteuttaa AppTiming-rajapinnan ja jonka palveluita käyttävät kaikki peli- tai delta-aikaa käyttävät komponentit.

5.1 Ohjelmakirjastot

Ohjelmakirjastot ovat valmiita koodikokoelmia, jotka sisältävät funktioita, luokkia, algoritmeja ja/tai ohjelmia. Pelimoottori voi sisältää useita erilaisia ohjelmakirjastoja, joita voidaan käyttää komponenttien ja pelimoottorilla kehitettävien pelien ohjelmointiin sekä pelimoottorin toiminnallisuuden ja ominaisuuksien monipuolistamiseen. Insinöörityössä toteutettu pelimoottori käyttää ohjelmakirjastoja grafiikan renderöintiin, fysiikan simulointiin sekä sovellusikkunan ja järjestelmätapahtumien hallintaan. [2, s. 31.]

5.1.1 Fysiikkakirjasto

Fysiikkakirjasto on ohjelmakirjasto, joka on tarkoitettu törmäysten havaitsemiseen ja jäykän kappaleen dynamiikkaan. Fysiikkakirjasto vastaa jäykän kappaleen kinematiikasta ja sen aiheuttamista voimista ja momenteista. Fysiikkakirjasto voi sisältää myös ajoneuvofysiikan, hahmomallinnuksen ja tuhoutuvan ympäristön mallinnuksen. [2, s. 602.]

Bullet on avoimen lähdekoodin törmäys- ja fysiikkakirjasto, joka on suunniteltu ensisijaisesti peli- ja elokuvatuotantoon sekä virtuaalisiin simulaatioihin. Bullet-fysiikkakirjaston törmäysmoottori on integroitu dynamiikan simulointiin, mutta törmäysjärjestelmää voidaan käyttää itsenäisenä tai muiden moottoreiden kanssa. Bullet-fysiikkakirjastoa käytetään insinöörityössä dynamiikan simulointiin ja törmäyksen tunnistukseen. [2, s. 602.]

Bullet-fysiikkakirjaston muokattava ja modulaarinen arkkitehtuuri on järjestetty yhdessä tai erikseen käytettäviin komponentteihin. Fysiikkakirjastosta on yhden tai kaksinkertaisen tarkkuuden versiot, joista voidaan käyttää valittuja osia tai laajentaa kirjastoja uusilla osilla. Bullet tukee muistin allokointia, profilointia ja virheenjäljitystä. [8, s. 6.]

Bullet-fysiikkakirjaston tärkeimmät ominaisuudet ovat diskreetti ja jatkuva törmäystarkistus, 6-vapausasteen jäykät kappaleet, yhdisteet ja nivelet, ajoneuvodynamiikka ja räsynukkesimulointi. Fysiikkakirjaston ominaisuuksia kuuluu myös pehmeän kehon dynamiikka kankaalle, köydelle ja muokattaville tilavuuksille. [8, s. 4.]

5.1.2 Grafiikkakirjasto

Grafiikkakirjasto on ohjelmakirjasto, joka on tarkoitettu grafiikan renderöintiin sekä renderöinnissä käytettävien varjostimien ja dynaamisen valaistuksen hallintaan. Grafiikkakirjasto tarjoaa yleensä näkymäalueen abstraktion ja siihen liittyvän kameramatriisin ja projektioparametrit, kuten näkökentän sekä lähi- ja kaukopiirtureiden sijainnit. [2, s. 37.]

OpenGL on peleissä, visualisoinneissa ja simulaatioissa käytetty järjestelmäriippumaton grafiikkakirjasto ja ohjelmointirajapinta vektorigrafiikan renderöintiin. Grafiikkakirjasto sisältää useita funktioita, joita voidaan käyttää monimutkaisen grafiikan, animaatioiden ja yksinkertaisten muotojen piirtämiseen. OpenGL tarjoaa korkeatasoisen varjostinohjelmointikielen pikseli- ja kulmapistevarjostimien ohjelmointiin. OpenGL-grafiikkakirjastoa käytetään insinöörityössä peligrafiikan piirtämiseen ja renderöintiin. [1, s. 261.]

OpenGL käyttää asiakkaaseen ja palvelimeen perustuvaa mallia, jossa ohjelman piirtokutsut välitetään asiakaan kautta palvelimelle. Grafiikkalaitteistossa, joka mahdollistaa grafiikkalaskennan suorittamisen rinnakkain, voidaan grafiikan työmäärä jakaa suorittimelle ja näytönohjaimelle, jossa suoritin toimii asiakkaana ja näytönohjain palvelimena. Asiakkaaseen ja palvelimeen perustuvassa mallissa etuna on, että asiakas voi palauttaa hallinnan sovellukseen ennen kuin komento on valmis. [9.]

5.1.3 Laitteistokirjasto

Laitteistokirjasto on ohjelmakirjasto, joka on tarkoitettu laitteistoresurssien ja järjestelmälaitteiden hallintaan. Laitteistokirjasto tarjoaa pääsyn järjestelmä-, grafiikka-, liitäntä- ja verkkolaitteiden järjestelmäresursseihin. Laitteistokirjastoa voidaan käyttää grafiikan renderöintiin, syötteiden lukemiseen ja multimedian toistamiseen. [1, s. 132.]

SDL on erityisesti peliohjelmointiin suunniteltu avoimen lähdekoodin monialustainen ohjelmakirjasto laitteistoresurssien hallintaan. SDL-ohjelmakirjastoa käytetään multimediaohjelmissa ja peleissä. SDL on kirjoitettu C-ohjelmointikielellä ja toimii suoraan C++-kielellä sekä linkitettynä myös useilla muilla ohjelmointikielillä. SDL-ohjelmakirjastoa käytetään insinöörityössä sovellusikkunan ja järjestelmätapahtumien hallintaan. [10.]

SDL-ohjelmakirjasto tarjoaa matalan tason pääsyn järjestelmä- ja grafiikkalaitteiden resursseihin OpenGL:n ja Direct3D:n kautta. SDL-ohjelmointirajapinta tukee laitteistokiihdytetyn grafiikan renderöintia, sovellusikkunoiden hallintaa, sovellus- ja ikkunatapahtumia, ohjainlaitteita ja kosketusnäyttöjä, äänentoistoa ja monikanavaääntä, tiedostojenkäsittelyä, rinnakkaisuutta, ajastimia ja laskureita, suorittimen- ja virranhallintaa. [10.]

5.2 Fysiikkamoottori

Fysiikkamoottori on jäykän kappaleen dynamiikan simuloinnista ja törmäyksen tunnistuksesta vastaava ydinkomponentti, jonka toteutuksesta vastaa neljä Bullet-fysiikkakirjastolla toteutettua fysiikkakomponenttia, jotka toteuttavat fysiikkapalveluiden fysiikkarajapinnat. Fysiikkarajapinnat mahdollistavat fysiikkamoottorin vaihtamisen toteutukseltaan erilaiseen fysiikkamoottoriin. Esimerkiksi Bullet-fysiikkakirjastolla toteutettu kolmiulotteisen fysiikan fysiikkamoottori voidaan vaihtaa Box2D-fysiikkakirjastolla toteutettuun kaksiulotteisen fysiikan fysiikkamoottoriin.

Fysiikkarajapinnat

Fysiikkarajapinta on fysiikkakomponenteissa toteutettavien palveluiden rajapinta, joka tarjoaa pääsyn fysiikkakomponentissa toteutettuihin palveluihin. PhysicsXD-fysiikkarajapinta (esimerkkikoodi 10) on fysiikan simuloinnista ja törmäyksien havaitsemisesta vastaavan kolmiulotteisen Physics3D-fysiikkakomponenttin rajapinta, jonka palveluita käyttävät jäykän kappaleen dynamiikasta vastaava Kinetics3D-fysiikkakomponentti sekä yhdisteistä ja nivelistä vastaava Compound3D-fysiikkakomponentti. ColliderXD-törmäytinrajapinta on törmäyttimistä vastaavan kolmiulotteisen Collider3D-fysiikkakomponenttin rajapinta, jonka palveluita käyttävät Physics3D-fysiikkakomponentti ja peliobjektien törmäyttiminä toimivat Collider-komponentit. KineticsXD-dynamiikkarajapinta on jäykän kappaleen dynamiikasta vastaavan kolmiulotteisen Kinetics3D-fysiikkakomponentin rajapinta, jonka palveluita käyttävät Physics3D-fysiikkakomponentti ja peliobjektien jäykänkappaleen dynamiikasta vastaavat RigidBody-komponentit. CompoundXD-yhdisterajapinta on peliobjektien yhdisteistä ja nivelistä vastaavan kolmiulotteisen Compound3D-fysiikkakomponentin rajapinta, jonka palveluita käyttävät Physics3D-fysiikkakomponentti ja peliobjektien yhdisteinä ja nivelinä toimivat Compound-komponentit.

```
class PhysicsXD {
public:
    virtual bool isContactResponse(const GameObject *aGameObject) = 0;
    virtual bool isTriggerResponse(const GameObject *aGameObject) = 0;
    virtual bool setGravity(const Vector3f &aGravity) = 0;
    virtual bool stateUpdate(bool aUpdate = true) = 0;
    // Some code omitted...
};

extern PhysicsXD *physicsXD;
```

Esimerkkikoodi 10. PhysicsXD-fysiikkarajapinta.

PhysicsXD-fysiikkarajapinta (esimerkkikoodi 10) on fysiikan simuloinnista ja törmäyksiä havaitsemisesta vastaavan fysiikkakomponentin rajapinta, joka tarjoaa pääsyn fysiikkakomponentissa toteutettuihin palveluihin. PhysicsXD-fysiikkarajapinta sisältää globaalin osoittimen ydinkomponentin palveluihin sekä virtuaaliset rajapintafunktiot törmäyksen tunnistukseen, painovoiman asettamiseen ja fysiikkamoottorin päivittämiseen.

5.3 Renderöintimoottori

Renderöintimoottori on peligrafiikan renderöinnista vastaava ydinkomponentti, jonka toteutuksesta vastaa OpenGL-grafiikkakirjastolla toteutettu OGraphics-ydinkomponentti, joka toteuttaa grafiikkapalveluiden grafiikkarajapinnan. Grafiikkarajapinta mahdollistaa renderöintimoottorin vaihtamisen toteutukseltaan erilaiseen renderöintimoottoriin. Esimerkiksi OpenGL-grafiikkakirjastolla toteutettu renderöintimoottori voidaan vaihtaa DirectX-grafiikkakirjastolla toteutettuun renderöintimoottoriin.

Grafiikkarajapinta

Grafiikkarajapinta on grafiikkakomponenteissa toteutettavien palveluiden rajapinta, joka tarjoaa pääsyn grafiikkakomponenteissa toteutettuihin palveluihin. XGraphics-grafiikkarajapinta (esimerkkikoodi 11) on peligrafiikkaa renderöivan OGraphics-ydinkomponentin rajapinta, jonka palveluita käyttävät peliobjektien Rendering-komponentit ja sovellusikkunan SdlWindow-ydinkomponentti sekä grafiikkasäteen RayCast-komponentti.

```
class XGraphics {
public:
    virtual bool doColouring(GameObject *aObject) = 0;
    virtual bool doTransform(GameObject *aObject) = 0;
    virtual bool doPrimitive(GameObject *aObject) = 0;
    virtual bool stateUpdate(bool aUpdate = true) = 0;
    virtual bool drawRayCast(RayCast *aRayCast) = 0;
    virtual bool clearGraphics() = 0;
    virtual bool renderObjects() = 0;
    // Some code omitted...
};

extern XGraphics *xGraphics;
```

Esimerkkikoodi 11. XGraphics-grafiikkarajapinta.

XGraphics-grafiikkarajapinta (esimerkkikoodi 11) on peligrafiikan renderöinnista vastaavan ydinkomponentin rajapinta, joka tarjoaa pääsyn ydinkomponentissa toteutettuihin palveluihin. XGraphics-grafiikkarajapinta sisältää globaalin osoittimen ydinkomponentin palveluihin sekä virtuaaliset rajapintafunktiot peliobjektien renderöintiin, grafiikkasäteen piirtämiseen ja renderöintimoottorin päivittämiseen.

5.4 Sovelluksenhallinta

Sovelluksenhallinta on sovelluksen ja sovellusikkunan hallinnasta vastaava sovelluksenhallintapalvelu, joka on toteutettu SDL-ohjelmakirjastolla toteutetuilla SdlEvents- ja SdlWindow-ydinkomponenteilla, jotka toteuttavat sovelluksenhallintapalveluiden sovelluksenhallintarajapinnat. Sovelluksenhallintarajapinnat mahdollistavat sovelluksenhallinnan vaihtamisen toteutukseltaan erilaiseen sovelluksenhallintaan. Esimerkiksi SDL-ohjelmakirjastolla toteutettu sovelluksenhallinta voidaan vaihtaa SFML-ohjelmakirjastolla toteutettuun sovelluksenhallintaan.

Ikkunointirajapinta

Ikkunointirajapinta on sovellusikkunakomponentissa toteutettavien palveluiden rajapinta, joka tarjoaa pääsyn sovellusikkunakomponentissa toteutettuihin palveluihin. AppWindow-Ikkunointirajapinta (esimerkkikoodi 12) on SdlWindow-ydinkomponentin rajapinta, jonka palveluita käyttää sovellusikkunaan renderöitavan peligrafiikan renderöinnista vastaava OGraphics-ydinkomponentti.

```
class AppWindow
{
public:
    virtual bool stateUpdate(bool aUpdate = true) = 0;
    virtual bool setWindow(const char* aWinName, const Vector2f aWinSite,
        const Vector2f aWinSize, int aWinMode) = 0;
    virtual Vector2f getWindowSize() = 0;
    virtual bool isActive() = 0;
    // Some code omitted...
};

extern AppWindow *appWindow;
```

Esimerkkikoodi 12. AppWindow-Ikkunointirajapinta.

AppWindow-Ikkunointirajapinta (esimerkkikoodi 12) on sovellusikkunan hallinnasta vastaavan ydinkomponentin rajapinta, joka tarjoaa pääsyn ydinkomponentissa toteutettuihin palveluihin. AppWindow-Ikkunointirajapinta sisältää globaalin osoittimen ydinkomponentin palveluihin sekä virtuaaliset rajapintafunktiot sovellusikkunan asettamiseen, hallintaan ja päivittämiseen.

5.5 Tapahtumienhallinta

Tapahtumienhallinta on sovellus- ja järjestelmätapahtumista vastaava ydinkomponentti, jonka toteutuksesta vastaa SDL-ohjelmakirjastolla toteutettu SdlEvents-ydinkomponentti, joka toteuttaa tapahtumapalveluiden tapahtumarajapinnan. Tapahtumarajapinta mahdollistaa tapahtumienhallinnan vaihtamisen toteutukseltaan erilaiseen tapahtumienhallintaan. Esimerkiksi SDL-ohjelmakirjastolla toteutettu tapahtumienhallinta voidaan vaihtaa SFML-ohjelmakirjastolla toteutettuun tapahtumienhallintaan.

Tapahtumarajapinta

Tapahtumarajapinta on tapahtumienhallintakomponentissa toteutettavien palveluiden rajapinta, joka tarjoaa pääsyn tapahtumienhallintakomponentissa toteutettuihin palveluihin. AppEvents-tapahtumarajapinta (esimerkkikoodi 13) on SdlEvents-ydinkomponentin rajapinta, jonka palveluita käyttävät sovellusikkunasta vastaava SdlWindow-ydinkomponentti ja syöttöjärjestelmästä vastaava AppInputs-komponentti.

```
class AppEvents
{
public:
    virtual bool getWindowEvents(WinEvents &aWinEvents) = 0;
    virtual bool getSystemEvents(SysEvents &aSysEvents) = 0;
    virtual bool stateUpdate(bool aUpdate = true) = 0;
    // Some code omitted...
};

extern AppEvents *appEvents;
```

Esimerkkikoodi 13. AppEvents-tapahtumarajapinta.

AppEvents-tapahtumarajapinta (esimerkkikoodi 13) on sovellus- ja järjestelmätapahtumista vastaavan ydinkomponentin rajapinta, joka tarjoaa pääsyn ydinkomponentissa toteutettuihin palveluihin. AppEvents-tapahtumarajapinta sisältää globaalin osoittimen ydinkomponentin palveluihin sekä virtuaaliset rajapintafunktiot sovellusikkuna- ja järjestelmätapahtumien hallintaan.

5.6 Reaaliaikajärjestelmä

Reaaliaikajärjestelmä on pelisilmukan ajoituksesta ja kehysnopeudesta vastaava ydinkomponentti, jonka toteutuksesta vastaa GameClock-ydinkomponentti, joka toteuttaa reaaliaikapalveluiden reaaliaikarajapinnan. Reaaliaikarajapinta mahdollistaa reaaliaikajärjestelmän vaihtamisen toteutukseltaan erilaiseen reaaliaikajärjestelmään.

Reaaliaikarajapinta

Reaaliaikarajapinta on reaaliaikakomponentissa toteutettavien palveluiden rajapinta, joka tarjoaa pääsyn reaaliaikakomponentissa toteutettuihin palveluihin. AppTiming-reaaliaikarajapinta (esimerkkikoodi 14) on GameClock-ydinkomponentin rajapinta, jonka palveluita käyttävät pelisilmukasta vastaava GameEngine-ydinkomponentti ja stateUpdate-funktion toteuttavat ydinkomponentit sekä kaikki delta-aikaa käyttävät komponentit.

```
class AppTiming
{
public:
    virtual bool maxFrameRate(int aFrameRate) = 0;
    virtual bool maxFixedRate(int aFixedRate) = 0;
    virtual bool stateUpdate(bool aActive = true) = 0;
    virtual int getFrameRate() = 0;
    virtual int getFrameStep() = 0;
    virtual int getFixedRate() = 0;
    virtual int getFixedStep() = 0;
    virtual float getDeltaTime() = 0;
    // Some code omitted...
};

extern AppTiming *appTiming;
```

Esimerkkikoodi 14. AppTiming-reaaliaikarajapinta.

AppTiming-reaaliaikarajapinta (esimerkkikoodi 13) on pelisilmukan ajoituksesta ja kehysnopeudesta vastaavan ydinkomponentin rajapinta, joka tarjoaa pääsyn ydinkomponentissa toteutettuihin palveluihin. AppTiming-reaaliaikarajapinta sisältää globaalin osoittimen ydinkomponentin palveluihin sekä virtuaaliset rajapintafunktiot pelisilmukan ajoituksen ja kehysnopeudesta hallintaan ja reaaliaikajärjestelmän päivittämiseen.

5.7 Näkymänhallinta

Näkymänhallinta on pelin toteuttamisesta ja hallinnasta vastaava ydinkomponentti, jonka toteutuksesta vastaa kolme näkymänhallintaluokkaa. Näkymänhallinta lataa ja käynnistää pelinäkymät sekä yhdistää peliresurssit ja ydinkomponenttien toiminnallisuuden pelattaviksi tapahtumiksi. Näkymänhallinta on näkymänhallintaluokkien hierarkkinen järjestelmä, jossa pelinäkymien hallinnasta vastaava Peliavaruus-luokka hallitsee pelitoteutuksien hallinnasta vastaavia Pelimaailma-luokkia, jotka hallitsevat pelitoteutuksien peliresursseista ja peliobjekteista vastaavia Pelitaso-luokkia, jotka mahdollistavat peliresursseihin ja peliobjekteihin hallinnan pelinäkymän ulkopuolelta.

5.7.1 Peliavaruus

Peliavaruusluokka (esimerkkikoodi 15) on näkymänhallinnan globaali luokka, joka hallitsee pelimaailmaluokasta perittyjä pelimaailmoja ja päivittää niiden tapahtumafunktiot (liite 1). Peliavaruusluokan dynaaminen muistinhallinta mahdollistaa pelimaailmoja ajonaikaisen lataamisen ja poistamisen sekä nopean vaihtamisen pelimaailmasta toiseen. Peliavaruusluokan tietorakenne sisältää kaikkien pelissä käytettävien pelimaailmojen osoittimet, joita voidaan lisätä ja poistaa peliavaruusluokan käsittelyfunktioilla (liite 1). Peliavaruusluokan toiminta perustuu dynaamiseen tietorakenteeseen sekä pelimaailmojen yksilölliseen indeksointiin.

```
class GameSpace : private SpaceTasks {
public:
    // Some code omitted...
    bool insertWorld(GameWorld *aWorld);
    bool verifyWorld(GameWorld *aWorld);
    bool removeWorld(GameWorld *aWorld);
    bool deleteWorld(GameWorld *aWorld);
    GameWorld* searchWorld(int aIdIndex);
    bool stateUpdate(bool aUpdate = true);
    bool removeWorlds();
    bool destroySpace();
};

extern GameSpace *gameSpace;
```

Esimerkkikoodi 15. Peliavaruusluokka.

5.7.2 Pelimaailmat

Pelimaailmaluokka (esimerkkikoodi 16) on näkymänhallinnan perittävä luokka, joka hallitsee pelitasoluokasta perittyjä pelitasoja ja päivittää niiden tapahtumafunktiot (liite 1). Pelimaailmaluokan dynaaminen muistinhallinta mahdollistaa pelitasoluokasta perittyjen pelitasojen ajonaikaisen lataamisen ja poistamisen sekä nopean vaihtamisen pelitasosta toiseen. Pelimaailman tietoraksenne sisältää pelimaailmassa käytettävien pelitasojen osoittimet, joita voidaan lisätä ja poistaa pelimaailmaluokan käsittelyfunktioilla (liite 1). Pelimaailmaluokan hallintafunktiot (liite 1) mahdollistavat pelimaailman suorituksen hallinnan ja palauttamisen pelimaailmaa ladattaessa. Pelimaailmaluokan virtuaaliset tapahtumafunktiot (liite 1) vastaavat käsittely- ja hallintafunktioiden tapahtumakutsuista sekä pelimaailman ja pelitasojen dynaamisesti päivittyvästä toiminnallisuudesta.

```
class GameWorld : private WorldTasks {
public:
    // Some code omitted...
    bool startWorld();
    bool awakeWorld();
    bool pauseWorld();
    bool closeWorld();
    bool resetWorld();
protected:
    bool insertLevel(GameLevel *aLevel);
    bool verifyLevel(GameLevel *aLevel);
    bool removeLevel(GameLevel *aLevel);
    bool deleteLevel(GameLevel *aLevel);
    GameLevel* searchLevel(int aIdIndex);
    virtual void startAction();
    virtual void awakeAction();
    virtual void pauseAction();
    virtual void closeAction();
    virtual void resetAction();
    virtual void frameUpdate();
    virtual void fixedUpdate();
    virtual void finalUpdate();
    virtual void insertEvent();
    virtual void removeEvent();
    virtual void deleteEvent();
};
```

Esimerkkikoodi 16. Pelimaailmaluokka.

5.7.3 Pelitasot

Pelitasoluokka (esimerkkikoodi 17) on perittävä luokka, joka hallitsee peliresursseja ja peliobjekti kantaluokasta perittyjä peliobjekteja sekä sisältää mahdollisuuden vaikuttaa pelitason tapahtumiin. Pelimaailmaluokan dynaaminen muistinhallinta mahdollistaa peliobjektien ajonaikaisen lataamisen ja poistamisen sekä uusien peliobjektien luomisen. Pelitason tietorakenne sisältää pelitasossa käytettävien peliobjektien osoittimet, joita voidaan lisätä ja poistaa pelitasoluokan käsittelyfunktioilla (liite 1). Pelitasoluokan hallintafunktiot (liite 1) mahdollistavat pelitason suorituksen hallinnan ja palauttamisen pelitasoa ladattaessa. Pelitasoluokan virtuaaliset tapahtumafunktiot (liite 1) vastaavat käsittely ja hallintafunktioiden tapahtumakutsuista sekä pelitason ja peliobjektien dynaamisesti päivittyvästä toiminnallisuudesta.

```
class GameLevel : private LevelTasks {
public:
    // Some code omitted...
    bool startLevel();
    bool awakeLevel();
    bool pauseLevel();
    bool closeLevel();
    bool resetLevel();
protected:
    bool insertObject(GameObject *aObject);
    bool removeObject(GameObject *aObject);
    bool deleteObject(GameObject *aObject);
    bool verifyObject(GameObject *aObject);
    GameObject* searchObject(int aIdIndex);
    GameObject* searchObject(char* aIdTag);
    virtual void startAction();
    virtual void awakeAction();
    virtual void pauseAction();
    virtual void closeAction();
    virtual void resetAction();
    virtual void frameUpdate();
    virtual void fixedUpdate();
    virtual void finalUpdate();
    virtual void insertEvent();
    virtual void removeEvent();
    virtual void deleteEvent();
};
```

Esimerkkikoodi 17. Pelitasoluokka.

5.7.4 Peliobjektit

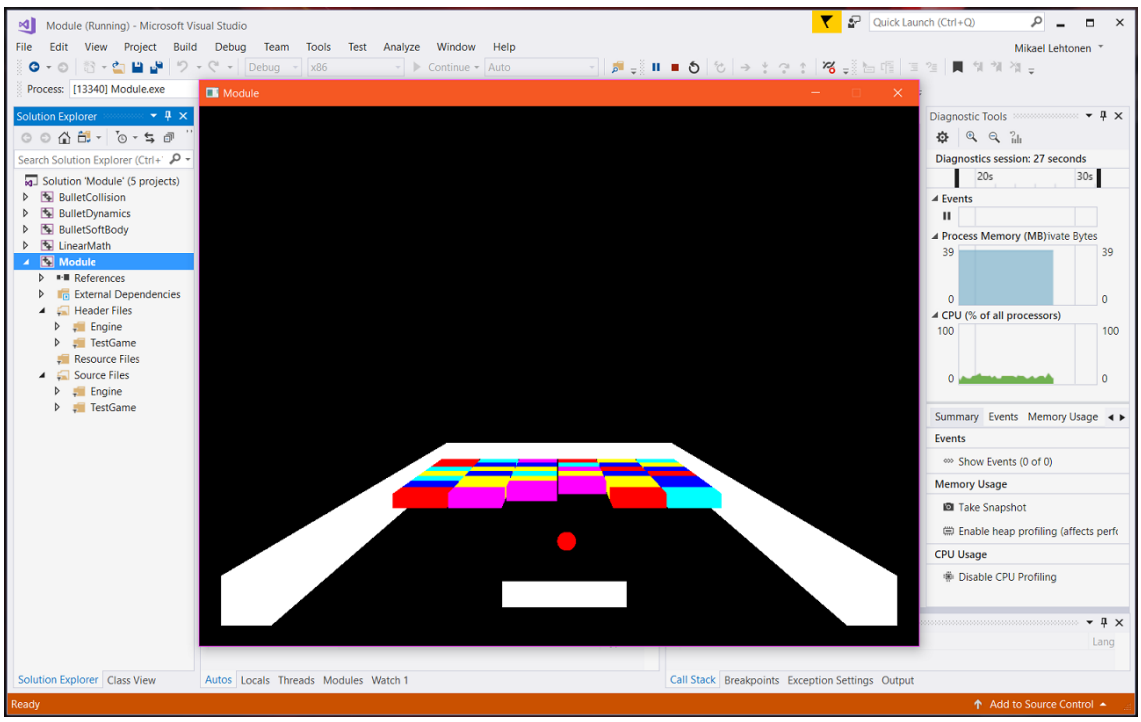
Peliobjektiluokka (esimerkkikoodi 18) on perittävä luokka, joka hallitsee komponenttikantaluokasta perittyjä komponentteja ja sisältää mahdollisuuden vaikuttaa peliobjektin käyttäytymiseen ja toiminnallisuuteen. Peliobjektiluokan dynaaminen muistinhallinta mahdollistaa komponenttien suorituksenaikaisen lisäämisen ja poistamisen. Peliobjektin tietorakenne sisältää peliobjektissa käytettävien komponenttien osoittimet, joita voidaan lisätä ja poistaa peliobjektin komponenttifunktioilla (liite 1). Peliobjektiluokan virtuaaliset tapahtumafunktiot (liite 1) vastaavat pelitasoluokan peliobjektinkäsittelyn tapahtumakutsuista sekä peliobjektien dynaamisesti päivittyvästä toiminnallisuudesta. Peliobjektiluokan fysiikkafunktiot (liite 1) vastaavat fysiikkamoottorin törmäysjärjestelmän tapahtumakutsuista. Törmäysjärjestelmä havaitsee törmäytinkomponentin sisältämien peliobjektien törmäykseen ja kontaktiin liittyvät tapahtumat ja kutsuu törmäyskomponentin sisältämien peliobjektien törmäyskontaktin ja -laukaisimen fysiikkafunktioita.

```
class GameObject {
public:
    // Some code omitted..
    virtual bool insertComponent(const Component *aComponent);
    virtual bool verifyComponent(const Component *aComponent);
    virtual bool removeComponent(const Component *aComponent);
    Component* searchComponent(int aIdIndex);
    Component* searchComponent(char* aIdTag);
protected:
    virtual void insertEvent();
    virtual void activeEvent();
    virtual void removeEvent();
    virtual void deleteEvent();
    virtual void fixedUpdate();
    virtual void frameUpdate();
    virtual void finalUpdate();
    virtual void contactCome(Contact aContact);
    virtual void contactStay(Contact aContact);
    virtual void contactExit(Contact aContact);
    virtual void triggerCome(Trigger aTrigger);
    virtual void triggerStay(Trigger aTrigger);
    virtual void triggerExit(Trigger aTrigger);
};
```

Esimerkkikoodi 18. Peliobjektiluokka.

6 Testaus ja analysointi

Insinöörityössä toteutetun pelimoottorin testaus ja analysointi suoritettiin pelimoottorilla toteutetulla testipelillä (kuva 4), jolla testattiin ydinkomponenttien ja näkymänhallinnan toimintaa sekä pelimoottorin suorituskykyä. Pelimoottori ja pelimoottorin testaukseen käytetty testipeli toteutettiin Microsoft Visual Studio ohjelmankehitysympäristöllä, jolla suoritettiin myös pelimoottorin suoritin- ja muistinkäytön analysoinnit (liite 2). Analysoinnit suoritettiin Visual Studion suorituskyvynprofilointi työkalulla, jolla voidaan analysoida sovelluksessa suoritettavien funktioiden suoritinaikaa ja muistinkäyttöä. Testauksen ja analysointien perusteella voidaan todeta insinöörityölle asetettujen tavoitteiden täyttyneen. Pelimoottoriin suunniteltuja ydinkomponentteja on toteuttamatta, joten suorituskyvyn lopullista optimointia ei ole vielä suoritettu, joka voidaan havaita suoritin- ja muistinkäytön analysoinneista (liite 2). Suurimmaksi suorittimen- ja muistinkäyttöön vaikuttavaksi tekijäksi voidaan analysointien perusteella todeta Bullet-fysiikkamoottori ja fysiikkamoottorin palveluita käyttävät peliobjektit. Muistinkäyttöön vaikuttavat eniten peliobjektien fysiikkakomponentit, jotka vastaavat peliobjektien dynamiikan simuloinnista ja törmäyksen tunnistuksesta. Ydinkomponenteista suoritinaikaa käyttää eniten pelimoottoriluokan pelisilmukkafunktio, joka vastaa pelitapahtumien päivittämisestä ja peligrafiikan renderöinnista sekä pelaajan antamien syötteiden havaitsemisesta.



Kuva 4. Pelimoottorin testipeli.

7 Yhteenveto

Insinöörityössä perehdyttiin modernin pelimoottorin historiaan ja merkitykseen pelituotannossa sekä komponenttipohjaisen pelimoottorin arkkitehtuuriin ja suunnitteluun. Historiassa tutustuttiin modernin pelimoottorin varhaisiin versioihin sekä pelimoottorien lisensoinnin vaikutuksesta kaupallisten pelimoottorin syntyyn. Pelituotannosta käsiteltiin valmiiden kaupallisten ja avoimen lähdekoodin pelimoottorien versioita ja pelimoottorien hyödyntämistä pelituotannossa. Arkkitehtuurissa selvitettiin komponenttipohjaisen pelimoottorin rakennetta ja toimintaa sekä pelimoottorin reaaliaikajärjestelmän ja pelisilmukan toteutusta. Suunnittelussa havainnollistettiin suunnitteluperiaatteiden ja suunnittelumallien hyödyntämistä komponenttipohjaisten pelimoottorien toteutuksessa.

Insinöörityössä toteutettiin modulaarinen komponenttipohjainen pelimoottori, joka sisältää toimivan pelin toteuttamiseen tarvittavat ohjelmistokomponentit. Pelimoottori ja ohjelmistokomponentit toteutettiin C++-ohjelmointikielellä sekä ohjelmointirajapintoja ja ohjelmakirjastoja hyödyntäen. Pelimoottorin modulaarinen rakenne mahdollistaa pelimoottorin ominaisuuksien laajentamisen uusilla komponenteilla ja toteutettujen komponenttien vaihtamisen ominaisuuksiltaan samankaltaisiin komponentteihin. Ohjelmistokomponenteista toteutettiin fysiikkaan, renderöintiin, sovelluksenhallintaan, reaaliaikajärjestelmään ja näkymänhallintaan liittyvät komponentit, joita voidaan täydentää resurssienhallintaan, äänenhallintaan ja skriptijärjestelmään tarvittavilla komponenteilla.

Insinöörityössä opittiin modernin komponenttipohjaisen pelimoottorin suunnittelussa ja toteuttamisessa tarvittavat tiedot ja taidot sekä C++-ohjelmointikielen ja ohjelmointirajapintojen hyödyntäminen pelimoottorin ohjelmoinnissa. Pelimoottorin arkkitehtuurista opittiin komponenttipohjaisen pelimoottorin ydinkomponenttien ja reaaliaikajärjestelmän merkitys pelimoottorin toteutuksessa sekä toteutuksessa tarvittavien ydinkomponenttien ominaisuudet ja reaaliaikajärjestelmän palvelut. Pelimoottorin suunnittelusta opittiin suunnitteluperiaatteiden ja pelimoottorin suunnittelumallien hyödyntäminen komponenttipohjaisen pelimoottorin toteutuksessa. Opittuja asioita voidaan hyödyntää pelimoottorien kehittämisessä ja ohjelmoinnissa sekä valmiiden pelimoottorien pelikehityksessä.

Lähteet

1 Alan Thorn. 2011. Game engine design and implementation. Jones & Bartlett Learning.

2 Jason Gregory. 2014. Game Engine Architecture, Second Edition. CRC Press.

3 List of game engines. 2017. Verkkodokumentti. Wikipedia. <https://en.wikipedia.org/wiki/List_of_game_engines>. Luettu 4.10.2017.

4 Open Source Game Development. 2012. Verkkodokumentti. Intel. <https://software.intel.com/en-us/articles/open-source-game-development>. Luettu 4.10.2017.

5 Game Programming Patterns / Sequencing Patterns / Game Loop. 2014. Verkkodokumentti. Robert Nystrom. <http://gameprogrammingpatterns.com/game-loop.html#intent>. Luettu 18.10.2017.

6 Game Programming Patterns / Sequencing Patterns / Update Method. 2014. Verkkodokumentti. Robert Nystrom. <http://gameprogrammingpatterns.com/game-loop.html#intent>. Luettu 19.10.2017.

7 Component Based Engine Design. 2013. Verkkodokumentti. Randy Gaul's Game Programming Blog. <http://www.randygaul.net/2013/05/20/component-based-engine-design/>. Luettu 2.10.2017.

8 Erwin Coumans. 2015. Bullet 2.83 Physics SDK Manual.

9 OpenGL Programming Guide for Mac. 2012. Verkkodokumentti. Apple. <https://developer.apple.com/library/content/documentation/GraphicsImaging/Conceptual/OpenGL-MacProgGuide/opengl_pg_concepts/opengl_pg_concepts.html>.Luettu 21.10.2017.

10 Simple DirectMedia Layer. 2017. Verkkodokumentti. <https://www.libsdl.org/> Luettu 22.10.2017.

Näkymänhallinnan funktiot

Taulukko 1. Peliavaruusluokan käsittelyfunktiot.

insertWorld	Lisää pelimaailman osoittimen peliavaruuden tietorakenteeseen ja kutsuu pelimaailman insertEvent-tapahtumafunktiota.
verifyWorld	Varmistaa pelimaailmojen osoittimia peliavaruuden tietorakenteesta.
removeWorld	Poistaa pelimaailman osoittimen peliavaruuden tietorakenteesta ja kutsuu poistettavan pelimaailman removeEvent-tapahtumafunktiota ja resetWorld-hallintafunktiota.
deleteWorld	Poistaa pelimaailman osoittimen peliavaruuden tietorakenteesta ja vapauttaa pelimaailman muistivarauksen sekä kutsuu pelimaailman deleteLevels-hallintafunktiota ja deleteEvent-tapahtumafunktiota.
searchWorld	Etsii pelimaailmojen osoittimia peliavaruuden tietorakenteesta.
stateUpdate	Päivittää peliavaruuden tapahtumat ja kutsuu pelimaailmojen frameUpdates-funktioita
removeWorlds	Poistaa pelimaailmojen osoittimet peliavaruuden tietorakenteesta ja kutsuu poistettavien pelimaailmojen removeEvent-tapahtumafunktioita ja resetWorld-hallintafunktioita.
destroySpace	Poistaa pelimaailmojen osoittimet peliavaruuden tietorakenteesta ja vapauttaa pelinmaa-ilmojen ja peliavaruuden muistivaraukset.

Taulukko 2. Pelimaailmaluokan käsittelyfunktiot.

insertLevel	Lisää pelitason osoittimen pelimaailman tietorakenteeseen ja kutsuu pelitason insertEvent-tapahtumafunktiota.
removeLevel	Poistaa pelitason osoittimen pelimaailman tietorakenteesta ja kutsuu poistettavan pelitason removeEvent-tapahtumafunktiota ja resetLevel-hallintafunktiota.
deleteLevel	Poistaa pelitason osoittimen tietorakenteesta ja vapauttaa pelitason muistivarauksen sekä kutsuu pelitason deleteObjects-hallintafunktiota ja deleteEvent-tapahtumafunktiota.
verifyLevel	Varmistaa pelitason osoittimen pelimaailman tietorakenteesta.
searchLevel	Etsii pelitason indeksejä pelimaailman tietorakenteesta.

Taulukko 3. Pelimaailmaluokan hallintafunktiot.

startWorld	Alustaa pelimaailman sekä asettaa pelimaailman peliavaruuden suoritettavaksi ja kutsuu pelimaailman startAction-tapahtumafunktiota.
pauseWorld	Keskeyttää pelitason suorituksen sekä kutsuu pelitason pauseLevel-hallintafunktiota ja pelimaailman pauseAction-tapahtumafunktiota.
awakeWorld	Asettaa pelimaailman peliavaruuden suoritettavaksi sekä kutsuu pelimaailman awakeAction-tapahtumafunktiota ja keskeytetyn pelitason awakeLevel-hallintafunktiota.
resetWorld	Pysäyttää käynnistetyt pelitasot sekä kutsuu pelimaailman resetAction-tapahtumafunktiota.
closeWorld	Pysäyttää käynnistetyt pelitasot sekä kutsuu pelitason closeLevel-hallintafunktiota ja pelimaailman closeAction-tapahtumafunktiota.

Taulukko 4. Pelimaailmaluokan tapahtumafunktiot.

startAction	Pelimaailman käynnistymisen vastaanottava tapahtumafunktio, jossa voidaan käynnistää pelimaailmassa suoritettavat tapahtumat.
awakeAction	Pelimaailman palauttamisen vastaanottava tapahtumafunktio, jossa voidaan palauttaa ja käynnistää pelimaailman keskeytetyt tapahtumat.
pauseAction	Pelimaailman keskeyttämisen vastaanottava tapahtumafunktio, jossa voidaan keskeyttää pelimaailmassa suoritettavat tapahtumat.
closeAction	Pelimaailman pysäyttämisen vastaanottava tapahtumafunktio, jossa voidaan pysäyttää pelimaailmassa suoritettavat tapahtumat.
resetAction	Pelimaailman suorituksen alustuksen vastaanottava tapahtumafunktio, jossa voidaan alustaa pelimaailmassa suoritettavat tapahtumat.
frameUpdate	Pelimaailman päivittymisen kehysnopeudella vastaanottava tapahtumafunktio, jossa voidaan päivittää pelimaailman tapahtumat.
fixedUpdate	Pelimaailman päivittyminen kiinteällä ajoituksella vastaanottava tapahtumafunktio, jossa voidaan päivittää fysiikkaan vaikuttavat tapahtumat.
finalUpdate	Pelimaailman päivittymisen vastaanottava tapahtumafunktio, joka suoritetaan pelisilmukan viimeisenä päivityksenä ja jossa voidaan päivittää grafiikkaan vaikuttavat tapahtumat.
insertEvent	Pelimaailman lisäyksen vastaanottava tapahtumafunktio, jossa voidaan suorittaa pelimaailman alustukseen liittyvät toimet.
removeEvent	Pelimaailman poistamisen vastaanottava tapahtumafunktio, jossa voidaan suorittaa pelimaailman poistamiseen liittyvät toimet.
deleteEvent	Pelimaailman tuhoamisen vastaanottava tapahtumafunktio, jossa voidaan suorittaa muistivarausten poistamiseen liittyvät toimet.

Taulukko 5. Pelitasoluokan käsittelyfunktiot.

insertObject	Lisää peliobjektin osoittimen pelitason tietorakenteeseen ja kutsuu peliobjektin insertEvent-tapahtumafunktiota.
removeObject	Poistaa peliobjektin osoittimen pelitason tietorakenteesta ja kutsuu poistettavan peliobjektin removeEvent-tapahtumafunktiota ja resetObject-hallintafunktiota.
deleteObject	Poistaa peliobjektin osoittimen tietorakenteesta ja vapauttaa peliobjektin muistivarauksen sekä kutsuu peliobjektin deleteObject-hallintafunktiota ja deleteEvent-tapahtumafunktiota.
verifyObject	Varmistaa peliobjektin osoittimen pelitason tietorakenteesta.
searchObject	Etsii peliobjektin tunnisteita pelitason tietorakenteesta.

Taulukko 6. Pelitasoluokan hallintafunktiot.

startLevel	Alustaa pelitason ja peliobjektit sekä asettaa pelitason pelimaailman suoritettavaksi ja kutsuu pelitason startAction-tapahtumafunktiota.
pauseLevel	keskeyttää pelitason suorituksen sekä kutsuu pelitason pauseAction-tapahtumafunktiota.
awakeLevel	Asettaa pelitason pelimaailman suoritettavaksi sekä kutsuu pelitason awakeAction-tapahtumafunktiota.
resetLevel	Pysäyttää pelitason suorituksen ja poistaa suoritettavat peliobjektit sekä kutsuu pelitason resetAction-tapahtumafunktiota.
closeLevel	Pysäyttää pelitason suorituksen sekä kutsuu pelitason closeAction-tapahtumafunktiota.

Taulukko 7. Pelitasoluokan tapahtumafunktiot.

startAction	Pelitason käynnistymisen vastaanottava tapahtumafunktio, jossa voidaan käynnistää pelitasossa suoritettavat tapahtumat.
awakeAction	Pelitason palauttamisen vastaanottava tapahtumafunktio, jossa voidaan palauttaa ja käynnistää pelitasossa keskeytetyt tapahtumat.
pauseAction	Pelitason keskeyttämisen vastaanottava tapahtumafunktio, jossa voidaan keskeyttää pelitasossa suoritettavat tapahtumat.
closeAction	Pelitason pysäyttämisen vastaanottava tapahtumafunktio, jossa voidaan pysäyttää pelitasossa suoritettavat tapahtumat.
resetAction	Pelitason suorituksen alustuksen vastaanottava tapahtumafunktio, jossa voidaan alustaa pelitasossa suoritettavat tapahtumat.
frameUpdate	Pelitason päivittymisen kehysnopeudella vastaanottava tapahtumafunktio, jossa voidaan päivittää pelitasossa tapahtumat.
fixedUpdate	Pelitason päivittyminen kiinteällä ajoituksella vastaanottava tapahtumafunktio, jossa voidaan päivittää fysiikkaan vaikuttavat tapahtumat.
finalUpdate	Pelitason päivittymisen vastaanottava tapahtumafunktio, joka suoritetaan pelisilmukan viimeisenä päivityksenä ja jossa voidaan päivittää grafiikkaan vaikuttavat tapahtumat.
insertEvent	Pelitason lisäyksen vastaanottava tapahtumafunktio, jossa voidaan suorittaa pelitasossa alustukseen liittyvät toimet.
removeEvent	Pelitason poistamisen vastaanottava tapahtumafunktio, jossa voidaan suorittaa pelitasossa poistamiseen liittyvät toimet.
deleteEvent	Pelitason tuhoamisen vastaanottava tapahtumafunktio, jossa voidaan suorittaa muistivarausten poistamiseen liittyvät toimet.

Taulukko 8. Peliobjektiluokan käsittelyfunktiot.

insertComponent	Lisää komponentin osoittimen peliobjektin tietorakenteeseen.
verifyComponent	Varmistaa komponentin osoittimen peliobjektin tietorakenteesta.
removeComponent	Poistaa komponentin osoittimen peliobjektin tietorakenteesta.
searchComponent	Etsii komponenttien tunnisteita peliobjektin tietorakenteesta.

Taulukko 9. Peliobjektiluokan tapahtumafunktiot.

insertEvent	Peliobjektin lisäyksen vastaanottava tapahtumafunktio, jossa voidaan suorittaa peliobjektin alustukseen liittyvät toimet.
actionEvent	Peliobjektin aktivoitumisen vastaanottava tapahtumafunktio, jossa voidaan suorittaa peliobjektin aktivoitumiseen liittyvät toimet.
removeEvent	Peliobjektin poistamisen vastaanottava tapahtumafunktio, jossa voidaan suorittaa peliobjektin poistamiseen liittyvät toimet.
deleteEvent	Pelimaailman tuhoamisen vastaanottava tapahtumafunktio, jossa voidaan suorittaa muistivarausten poistamiseen liittyvät toimet.
frameUpdate	Peliobjektin päivittymisen kehysnopeudella vastaanottava tapahtumafunktio, jossa voidaan päivittää peliobjektin tapahtumat.
fixedUpdate	Peliobjektin päivittyminen kiinteällä ajoituksella vastaanottava tapahtumafunktio, jossa voidaan päivittää fysiikkaan vaikuttavat tapahtumat.
finalUpdate	Peliobjektin päivittymisen vastaanottava tapahtumafunktio, joka suoritetaan pelisilmukan viimeisenä päivityksenä ja jossa voidaan päivittää grafiikkaan vaikuttavat tapahtumat.

Taulukko 10. Peliobjektiluokan fysiikkafunktiot.

contactCome	Törmäyskontaktin tapahtumisen vastaanottava fysiikkafunktio, jossa voidaan suorittaa peliobjektien törmäykseen liittyvät toimet.
contactStay	Törmäyskontaktin pysymisen vastaanottava fysiikkafunktio, jossa voidaan suorittaa peliobjektien kontaktiin liittyvät toimet.
contactExit	Törmäyskontaktin irtoaminen vastaanottava fysiikkafunktio, jossa voidaan suorittaa peliobjektien irtoamiseen liittyvät toimet.
triggerCome	Törmäyslaukaisimen tapahtumisen vastaanottava fysiikkafunktio, jossa voidaan suorittaa törmäykseen havaitsemiseen liittyvät toimet.
triggerStay	Törmäyslaukaisimen pysymisen vastaanottava fysiikkafunktio, jossa voidaan suorittaa kontaktin havaitsemiseen liittyvät toimet.
triggerExit	Törmäyslaukaisimen irtoamisen vastaanottava fysiikkafunktio, jossa voidaan suorittaa irtoamisen havaitsemiseen liittyvät toimet.

Suorituskyvyn analysoinnit

Taulukko 1. Suoritinkäytön analysointi.

Function Name	Total CPU (ms)	Total CPU (%)	Self CPU (ms)	Self CPU (%)
Module.exe (PID: 7056)	54531	100,00 %	0	0,00 %
kernel32.dll!0x000000779b8744	51991	95,34 %	0	0,00 %
ntdll.dll!0x00000077ad582d	51991	95,34 %	0	0,00 %
ntdll.dll!0x00000077ad57fd	51991	95,34 %	0	0,00 %
__scrt_common_main_seh	50927	93,39 %	0	0,00 %
__scrt_common_main	50927	93,39 %	0	0,00 %
mainCRTStartup	50927	93,39 %	0	0,00 %
SDL_main	50921	93,38 %	0	0,00 %
main_utf8	50921	93,38 %	0	0,00 %
main	50921	93,38 %	0	0,00 %
invoke_main	50921	93,38 %	0	0,00 %
GameEngine::GameLoop	50798	93,15 %	57	0,10 %
Physics3D::stateUpdate	30890	56,65 %	43	0,08 %
btDiscreteDynamicsWorld::	15693	28,78 %	34	0,06 %
Physics3D::updateObjects	14974	27,46 %	53	0,10 %
btDiscreteDynamicsWorld::	14421	26,45 %	15	0,03 %
Kinetics3D::isActiveRigidBody	11556	21,19 %	259	0,47 %
std::_Tree<std::_Tmap_traits<int,btRigid	11331	20,78 %	116	0,21 %
OGraphics::stateUpdate	8994	16,49 %	8	0,01 %
tickCallback	8974	16,46 %	3	0,01 %
Physics3D::tickCollisions	8950	16,41 %	2	0,00 %
OGraphics::renderObjects	8872	16,27 %	38	0,07 %
Physics3D::exitCollisions	8833	16,20 %	176	0,32 %
SdlWindow::stateUpdate	7497	13,75 %	47	0,09 %
std::_Tree<std::_Tmap_traits<int,btRigid	6871	12,60 %	74	0,14 %
SDL2.dll!0x0000000fe53aed	6286	11,53 %	5	0,01 %
SDL2.dll!0x0000000fe5ccf5	6260	11,48 %	1	0,00 %
std::_Tree<std::_Tmap_traits<int,Contact	6028	11,05 %	27	0,05 %
std::_Tree<std::_Tmap_traits<int,btRigid	5999	11,00 %	436	0,80 %
gdi32full.dll!0x00000076924eb6	5938	10,89 %	2	0,00 %
opengl32.dll!0x0000006904d547	5661	10,38 %	4	0,01 %
ig75icd32.dll!0x0000001040045b	5527	10,14 %	0	0,00 %
opengl32.dll!0x00000069040e3f	5527	10,14 %	0	0,00 %
OGraphics::doPrimitive	4601	8,44 %	33	0,06 %
OPrimitive::drawShape	4565	8,37 %	38	0,07 %
wow64.dll!0x00000058a4b130	4487	8,23 %	0	0,00 %
ntdll.dll!0x007ff971099b1b	4424	8,11 %	0	0,00 %
ntdll.dll!0x007ff971099ace	4424	8,11 %	0	0,00 %

Taulukko 2. Muistinkäytön analysointi.

Object Type	Count	Size (Bytes)
void	392	4 095 836
Module.exe!btRigidBody[]	48	33 936
Module.exe!btCompoundShape[]	144	18 864
Module.exe!std::_Tree_node<std::pair<int const ,Compound>,void *	48	18 624
Module.exe!TestObject	48	14 592
Module.exe!TestLevel	1	13 344
Module.exe!btDefaultMotionState[]	48	11 664
Module.exe!std::_Tree_node<std::pair<int const ,Collider>,void *>	96	10 368
Module.exe!btCollisionDispatcher	1	5 260
Module.exe!btBoxShape[]	47	4 653
Module.exe!std::_Tree_node<std::basic_string<char,std::char_trait	91	4 004
Module.exe!std::_Tree_node<std::pair<int const ,Contact>,void *>	16	2 240
Module.exe!std::_Container_proxy	258	2 064
Module.exe!std::_Tree_node<std::pair<int const ,btRigidBody *>,	49	1 176
Module.exe!std::_Tree_node<std::pair<int const ,btCompoundShap	49	1 176
Module.exe!std::_Tree_node<int,void *>	51	1 020
Module.exe!std::_Tree_node<std::pair<int const ,GameObject *>,	38	912
Module.exe!CProfileNode	20	720
Module.exe!std::_List_node<int,void *>	56	672
Module.exe!btDiscreteDynamicsWorld[]	1	355
Module.exe!btSequentialImpulseConstraintSolver[]	1	227
Module.exe!std::_Tree_node<std::pair<int const ,Vector3f>,void *>	4	208
Module.exe!btDbvtBroadphase	1	196
int	48	192
Module.exe!btDefaultCollisionConfiguration	1	92
Module.exe!btSphereShape[]	1	83
Module.exe!std::_Tree_node<std::pair<int const ,GameLevel *>,void	3	72
Module.exe!GameLevel	2	56
Module.exe!std::_Tree_node<std::pair<int const ,GameWorld *>,	2	48
Module.exe!TestWorld	1	36
Module.exe!GameWorld	1	32
Module.exe!btClockData	1	32
Module.exe!GameSpace	1	28
Module.exe!std::_Tree_node<std::pair<int const ,btTypedConstraint	1	24
Module.exe!std::_Tree_node<std::pair<int const ,_SDL_Joystick *>,	1	24
Module.exe!std::_Tree_node<std::pair<int const ,_SDL_GameContro	1	24